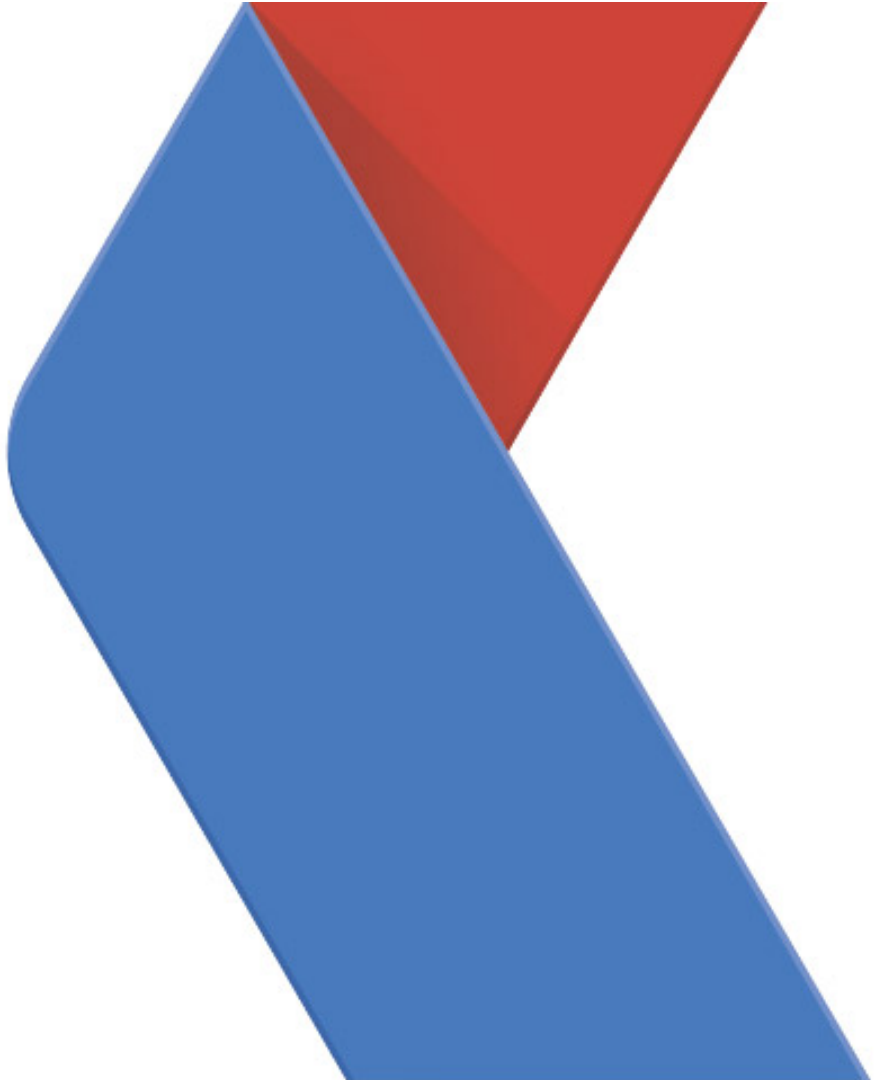
Vulkan: High-Performance 3D Graphics for Android

December 2016



About This Talk

Why a New Graphics API?

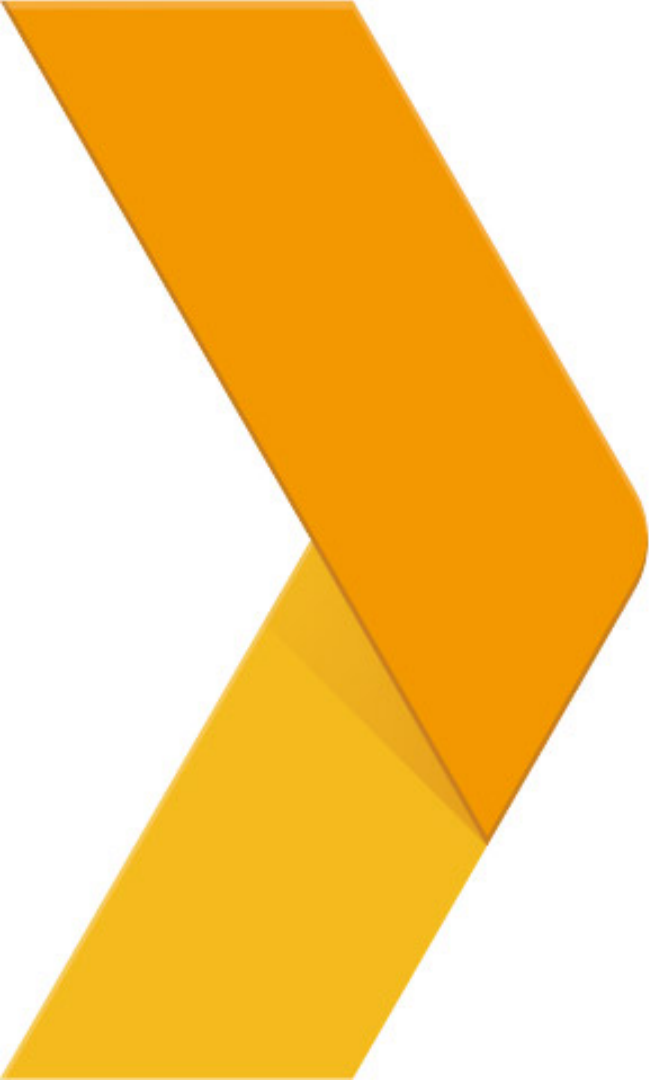
What is Vulkan?

Getting Started on Android

Tips and Best Practices for Mobile/Android

Is Vulkan Right for My App?

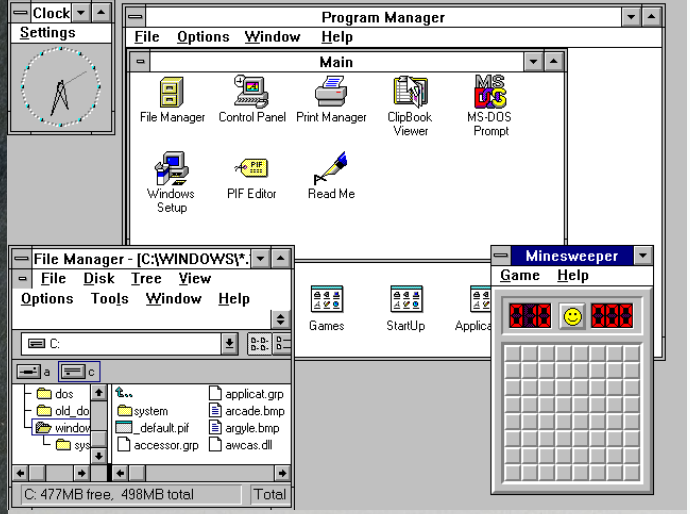
Google Developer Day



Starting Over

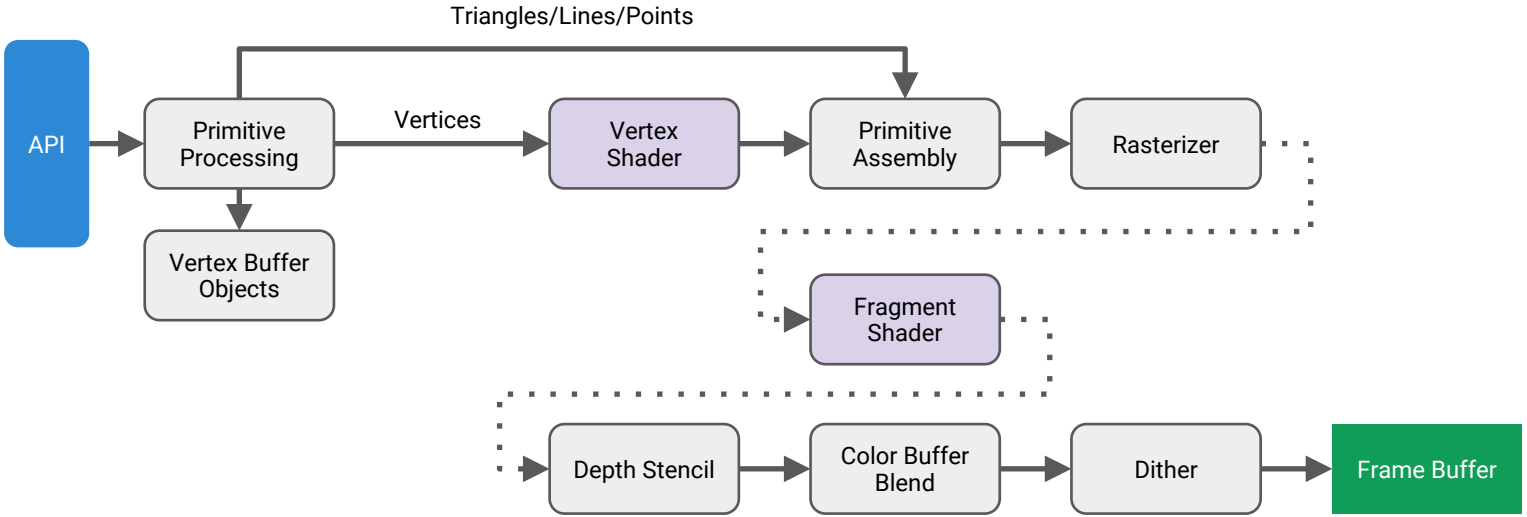
OpenGL

Released in 1992



Google Developer Day

OpenGL ES 2.0 Pipeline

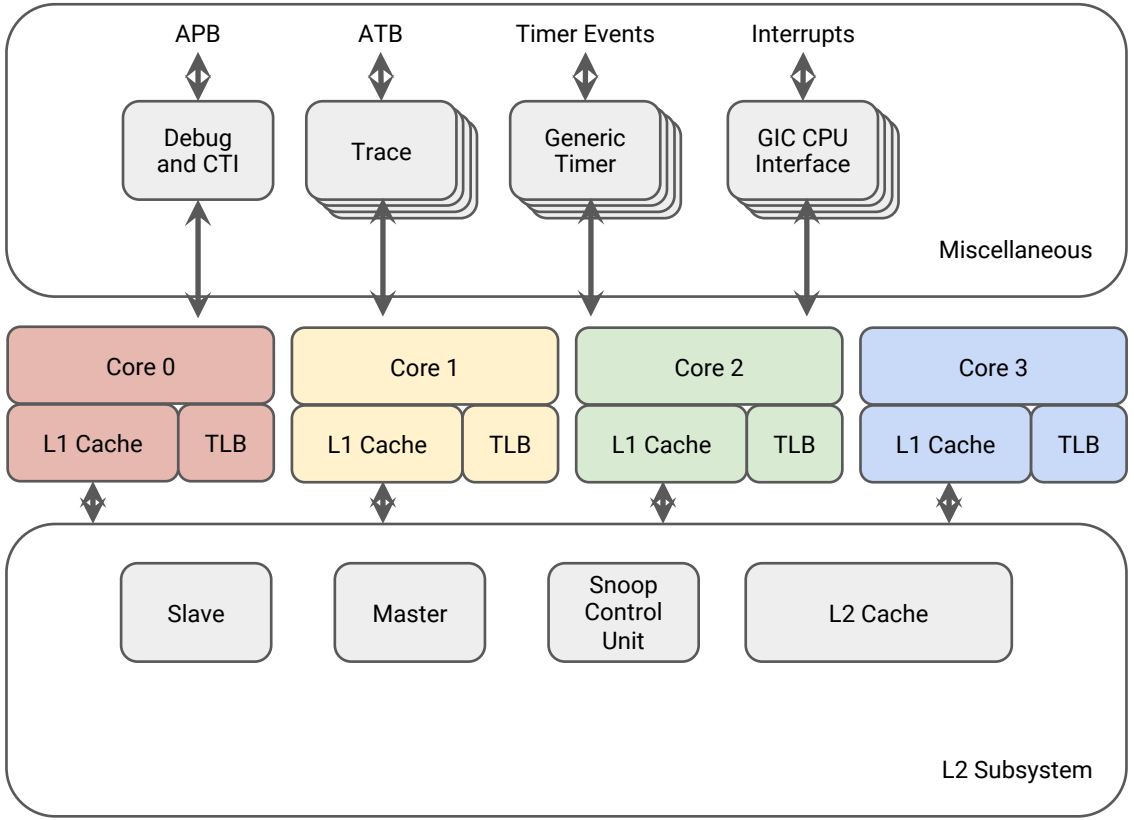


Mobile Computing Today

Multi-Core

> 2.0 GHz

Multi-Level Cache



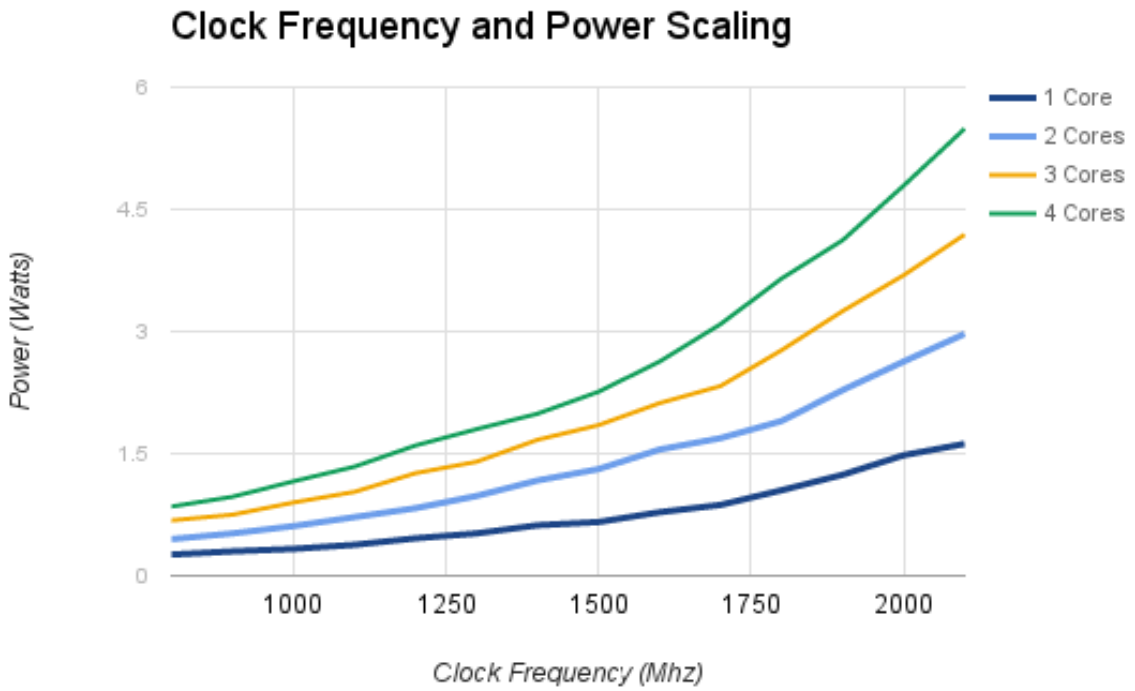
Power and Clock Speed

Multi-Core

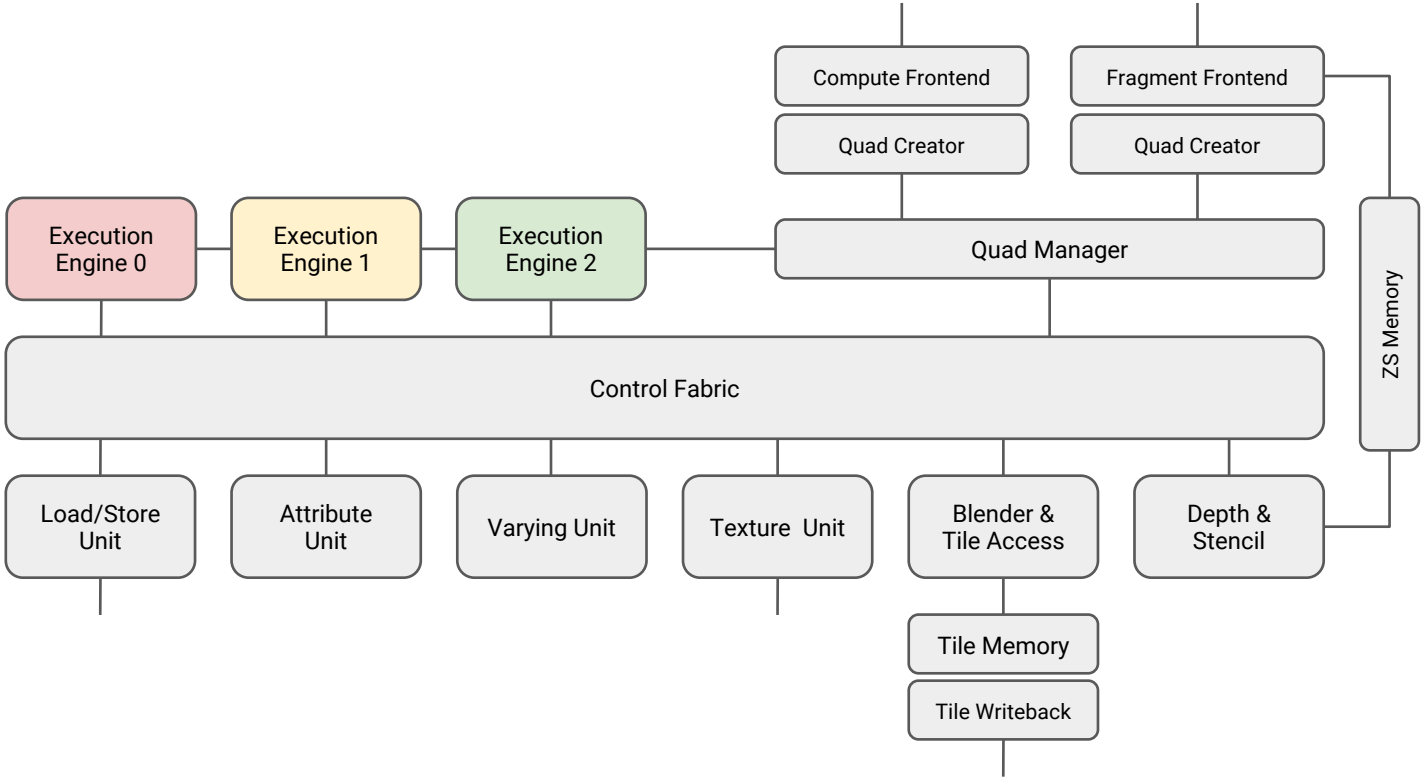
> 2.0 GHz

Multi-Level Cache

$P=CV^2f$



Modern GPU Design



Enter Vulkan

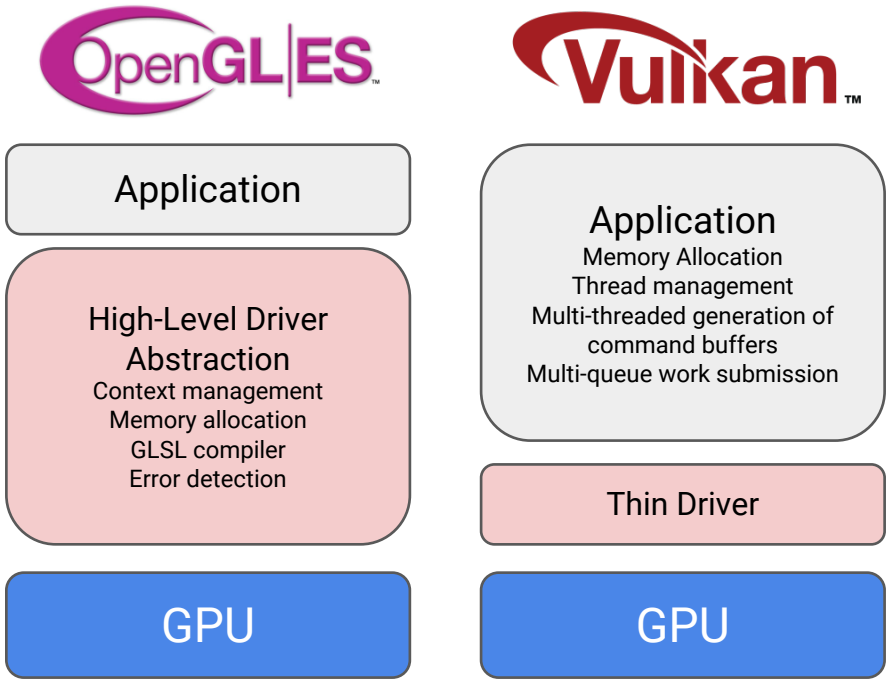
Vulkan

- Explicit
- Modern
- Extensible

Vulkan, an Explicit API

- Explicit
  - **Predictable behavior**
  - **Predictable performance**

Vulkan Explicit GPU Control



Vulkan, a Modern API

- Modern
  - **Designed for Desktop and Mobile**
  - Data-oriented
  - Multithreading
  - Shader IR (SPIR-V)

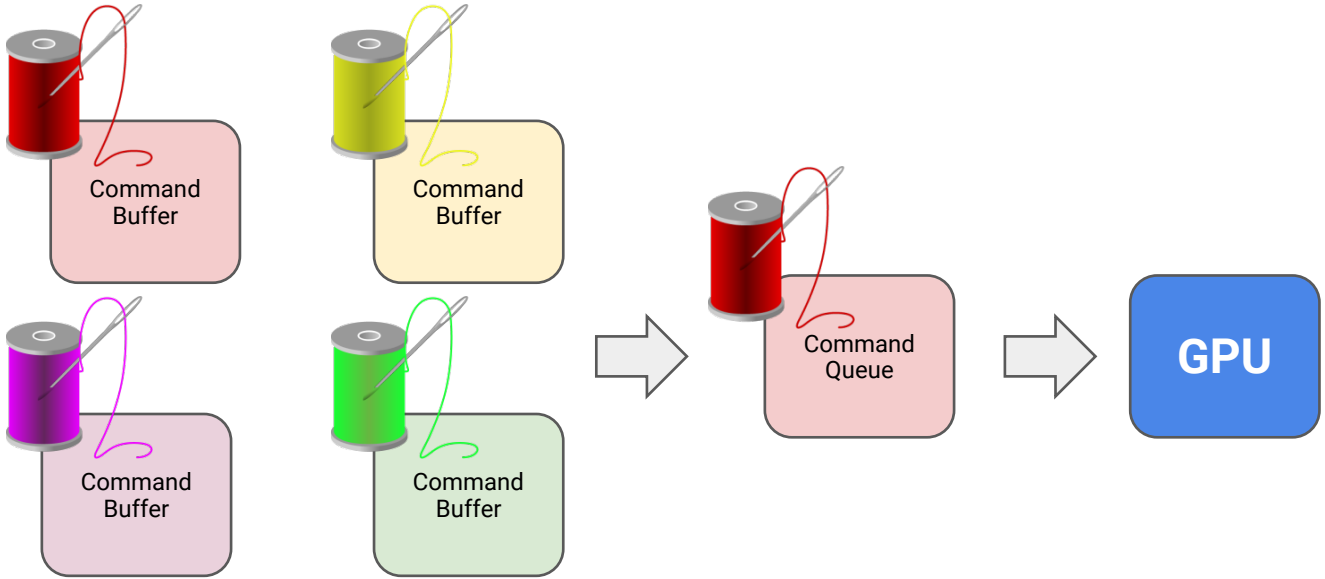
Vulkan, a Modern API

- Modern
  - Designed for Desktop and Mobile
  - **Data-oriented**
  - Multithreading
  - Graphics, Compute, DMA Queues
  - Shader IR (SPIR-V)

Vulkan, a Modern API

- Modern
  - Designed for Desktop and Mobile
  - Data-oriented
  - **Multithreading**
  - Graphics, Compute, DMA Queues
  - Shader IR (SPIR-V)

Vulkan Multithreading



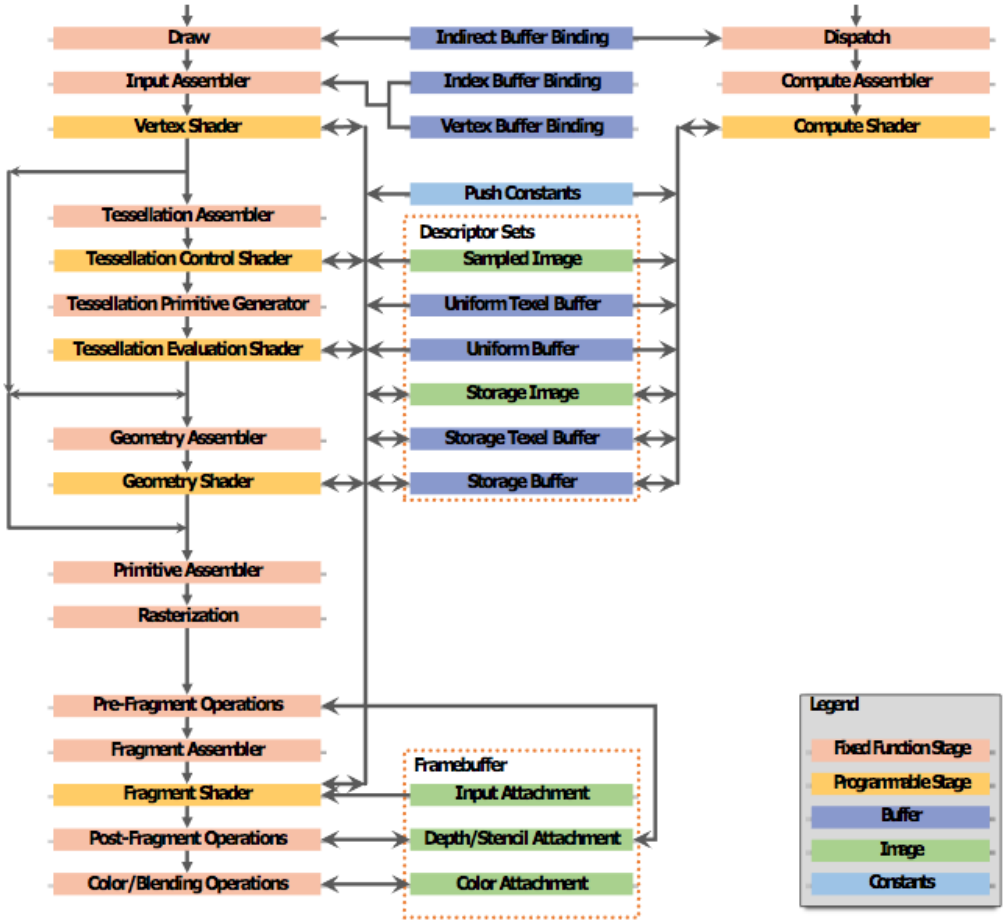
Vulkan, a Modern API

- Modern
  - Designed for Desktop and Mobile
  - Data-oriented
  - Multithreading
  - **Graphics, Compute, DMA Queues**
  - **Shader IR (SPIR-V)**

GLSL and HLSL are both supported by ShaderC!

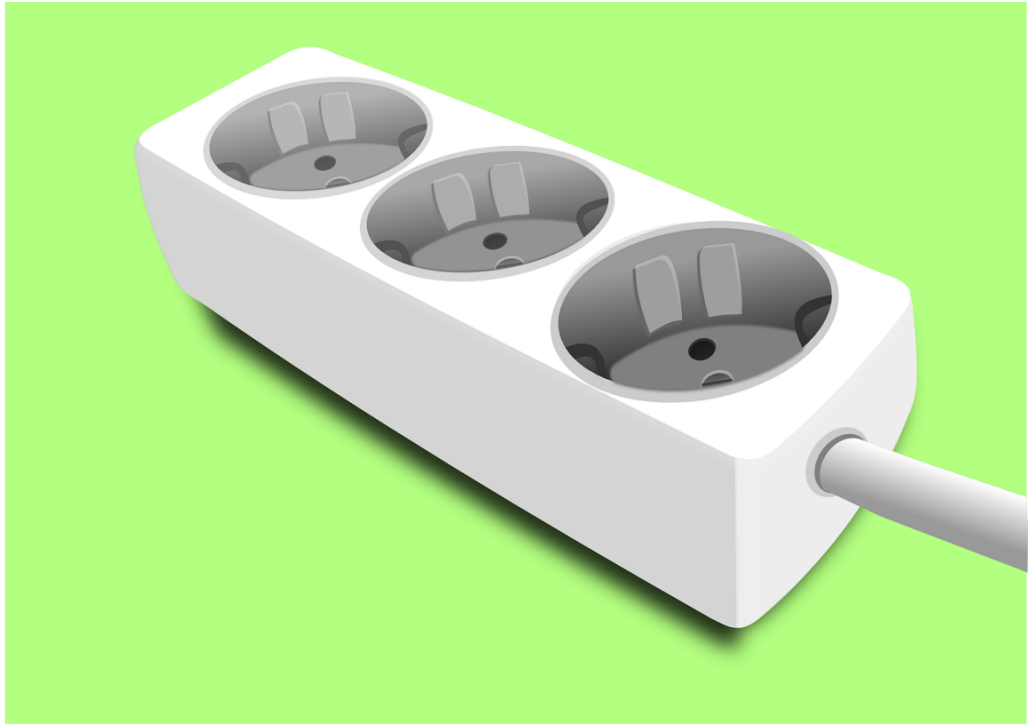
A Modern Pipeline

- Programmers are in the center of the pipeline
  - Design shaders
  - Configure fixed function units
  - Feed images/textures

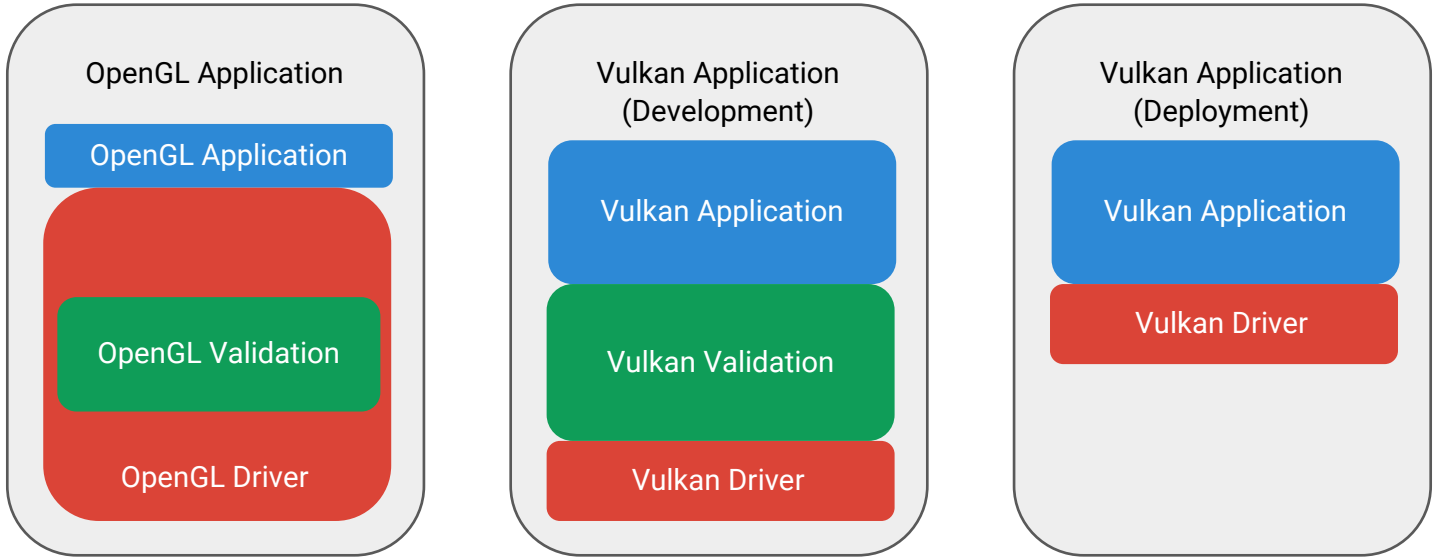


Vulkan, an Extensible API

- Extensible
  - **Layers and extensions**
  - Common loader interface

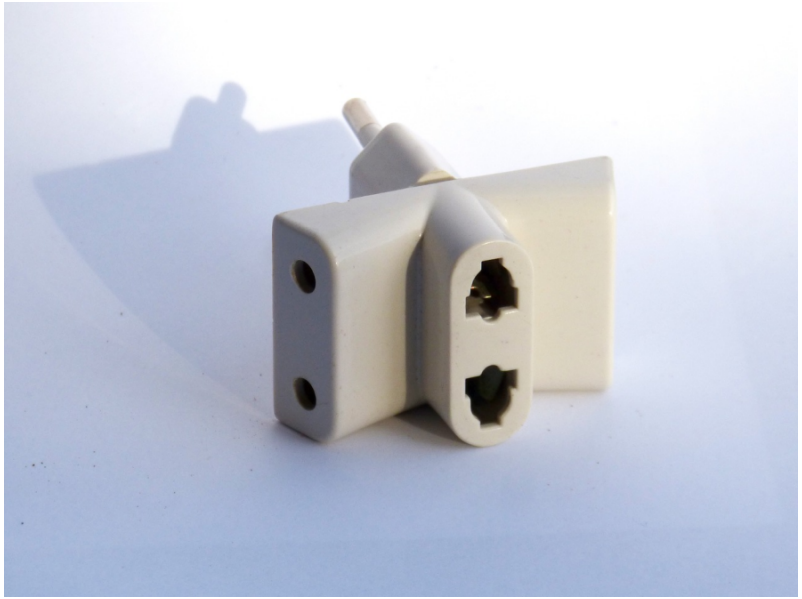


OpenGL and Vulkan



Vulkan, an Extensible API

- Extensible
  - Layers and extensions
  - **Common loader interface**



Vulkan, an Open Standard

- Open Source
  - Specification, Tests
  - Tools
  - Validation Layers
- Broad OS Support
  - Android
  - Windows 7,8, 10
  - Linux

Getting Started with Vulkan on Android

Some Vulkan Devices

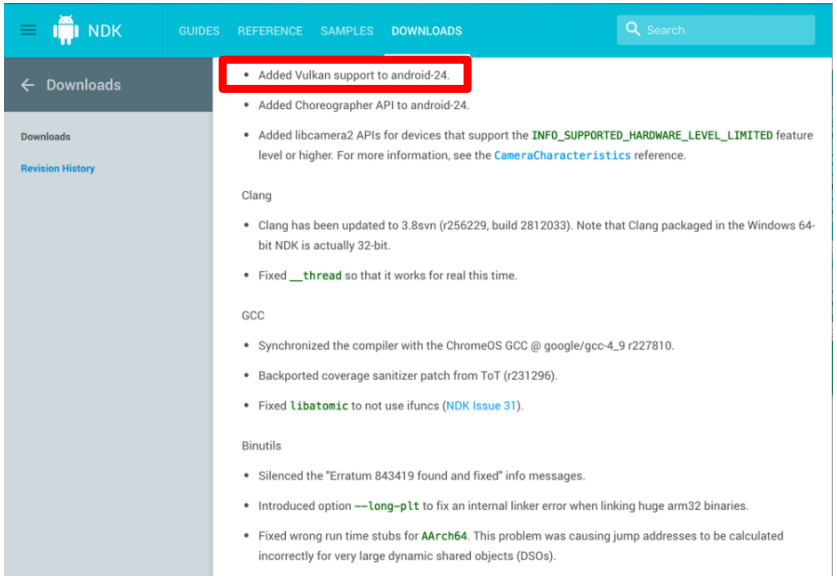
Google Developer Day

Android Nougat

d.android.com/about/versions/nougat

NDK Headers in R12

d.android.com/ndk/downloads



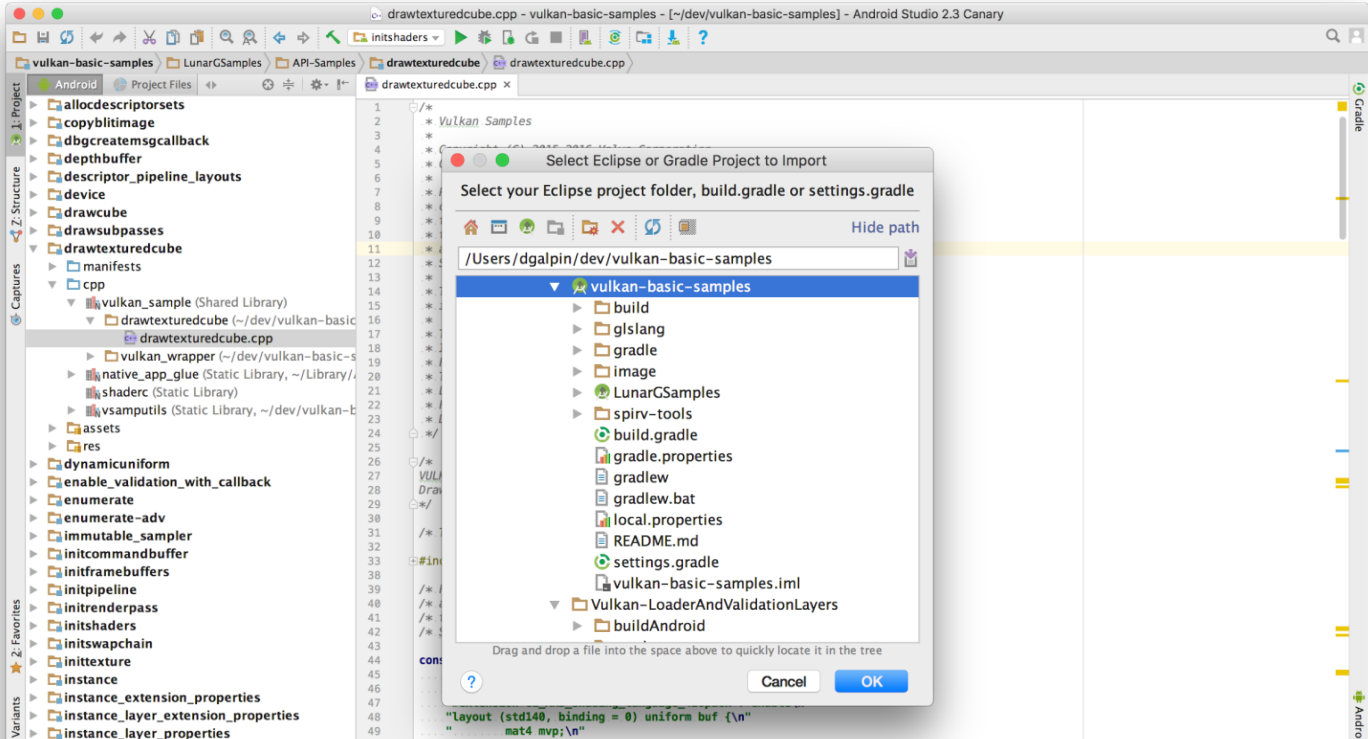
Vulkan Samples

github.com/googlesamples/vulkan-basic-samples.git

```sh
# Clone the repository
git clone https://github.com/googlesamples/vulkan-basic-samples.git

# Navigate to the LunarGSamples directory

# Update the glslang source
$ ./update_external_sources.sh -s -g

# Build the sample Android Studio projects
cd API-samples
$ cmake -DANDROID=ON -DANDROID_ABI=[armeabi-v7a|arm64-v8a|
x86|x86_64|all(default)]
```

Vulkan Samples in Android Studio

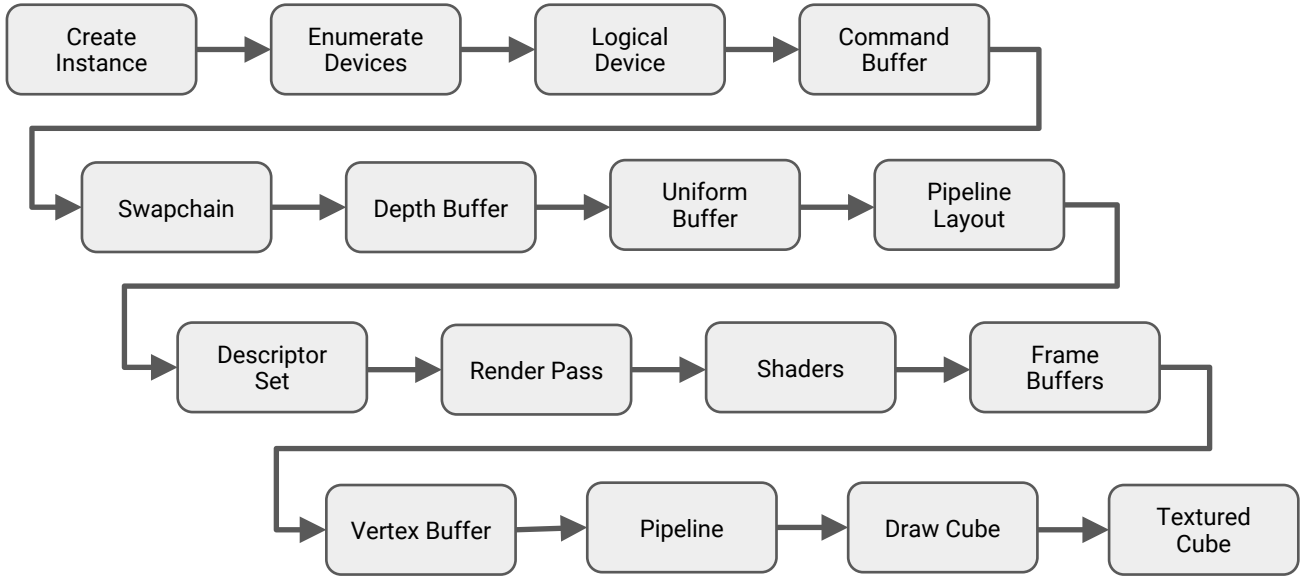
Open project at root of repo



Google Developer Day

Vulkan Samples in Android Studio

Sample Progression Tutorial (LunarXchange)



Tips for Developing in Vulkan

#1: Use Validation

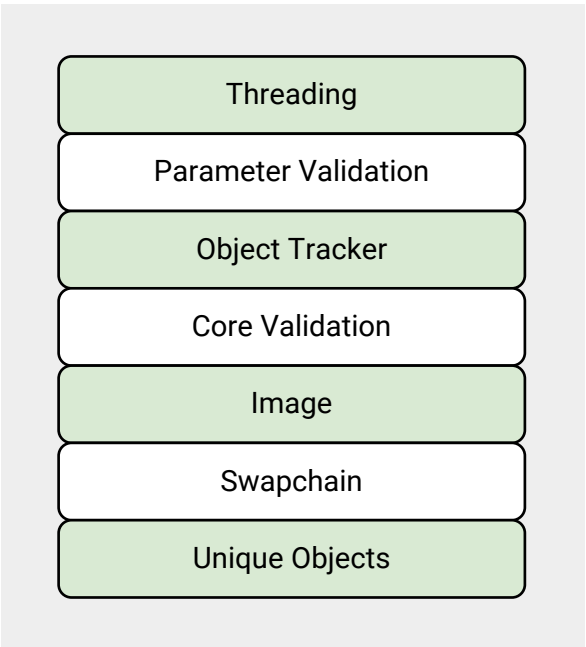
Vulkan Validation Layers

Your Defense Against Fragmentation

- Detects and reports incorrect API usage
- Capable of much more, including performance analysis
- Every early Vulkan developer tells us that using them is critical to their success

Vulkan Standard Validation Layers

```
const char *instance_layers[] = {
    "VK_LAYER_LUNARG_standard_validation"
};
```

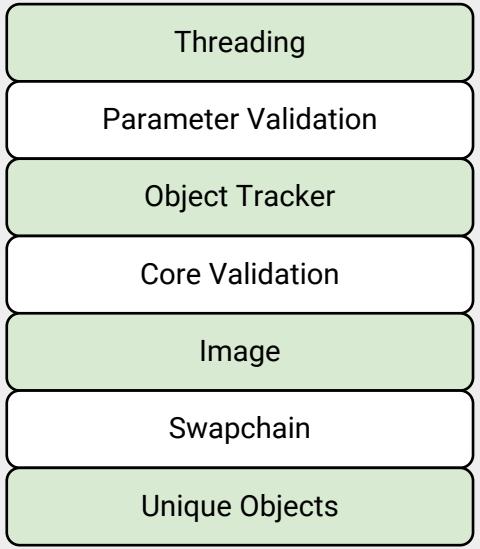


Vulkan Standard Validation Layers

LunarGSamples/layers

CMake

```
add_vk_layer(threading threading.cpp thread_check.h vk_layer_table.cpp)
add_vk_layer(parameter_validation parameter_validation.cpp parameter_validation.h
vk_layer_table.cpp)
add_vk_layer(object_tracker object_tracker.cpp vk_layer_table.cpp)
add_vk_layer(core_validation core_validation.cpp vk_layer_table.cpp)
add_vk_layer(image image.cpp vk_layer_table.cpp)
add_vk_layer(swapchain swapchain.cpp vk_layer_table.cpp)
add_vk_layer(unique_objects unique_objects.cpp vk_layer_table.cpp vk_safe_struct.cpp)
```

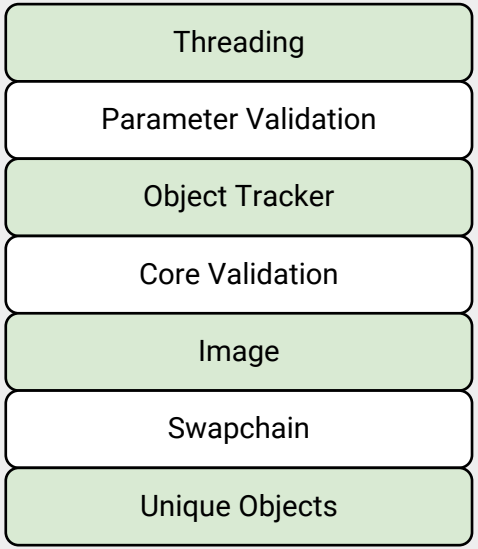


Vulkan Standard Validation Layers

LunarGSamples/layers

C++

```
const char *instance_layers[] = {
    "VK_LAYER_GOOGLE_threading",
    "VK_LAYER_LUNARG_parameter_validation",
    "VK_LAYER_LUNARG_object_tracker",
    "VK_LAYER_LUNARG_core_validation",
    "VK_LAYER_LUNARG_image",
    "VK_LAYER_LUNARG_swapchain",
    "VK_LAYER_GOOGLE_unique_objects"
};
```



Enabling Instance Validation

C++

```
const char *instance_layers[] = {
    "VK_LAYER_GOOGLE_threading",
    "VK_LAYER_LUNARG_parameter_validation",
    "VK_LAYER_LUNARG_object_tracker",
    "VK_LAYER_LUNARG_core_validation",
    "VK_LAYER_LUNARG_image",
    "VK_LAYER_LUNARG_swapchain",
    "VK_LAYER_GOOGLE_unique_objects"
};
uint32_t instance_layer_request_count = sizeof(instance_layers) / sizeof(instance_layers[0]);
// Pass desired instance layers into vkCreateInstance
VkInstanceCreateInfo instance_info = {};
instance_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
instance_info.enabledLayerCount = instance_layer_request_count;
instance_info.ppEnabledLayerNames = instance_layers;
```

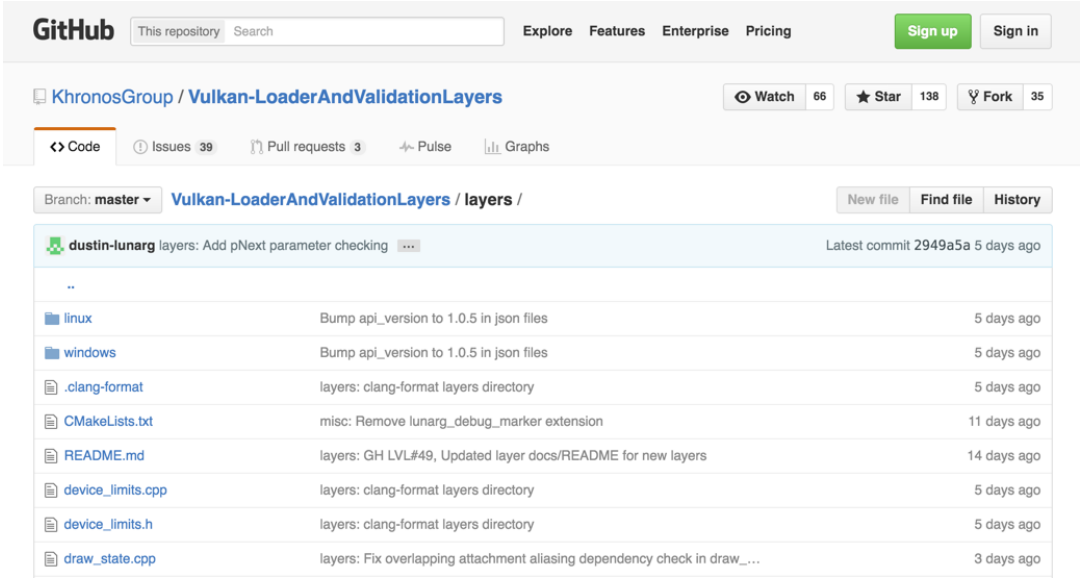
Enabling Instance Validation

C++

```
static VKAPI_ATTR VkBool32 VKAPI_CALL DebugReportCallback(
                          VkDebugReportFlagsEXT msgFlags,
                          VkDebugReportObjectTypeEXT objType,
                          uint64_t srcObject, size_t location,
                          int32_t msgCode, const char * pLayerPrefix,
                          const char * pMsg, void * pUserData )
{
  if (msgFlags & VK_DEBUG_REPORT_ERROR_BIT_EXT) {
     __android_log_print(ANDROID_LOG_ERROR,
                         "AppName",
                         "ERROR: [%s] Code %i : %s",
                         pLayerPrefix, msgCode, pMsg);
  } else if (msgFlags & VK_DEBUG_REPORT_WARNING_BIT_EXT) {
    ...
```

Vulkan Validation Layers

github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers/tree/master/layers



Building the Validation Layers

SH

```
# First clone the repository from GitHub (not shown)
# Then get the glslang peer directory

./update_external_sources.sh -g

# Generate the Android makefiles

cd build—android
sh android-generate.sh
ndk-build
```

Checking for Validation Layers (Instance)

C++

```
// Get instance layer count using null pointer as last parameter
uint32_t instance_layer_present_count = 0;
vkEnumerateInstanceLayerProperties(&instance_layer_present_count, nullptr);

// Enumerate instance layers with valid pointer in last parameter
VkLayerProperties* layer_props = malloc(instance_layer_present_count *
sizeof(VkLayerProperties));
vkEnumerateInstanceLayerProperties(instance_layer_present_count, layer_props));
```

Beyond Validation: Testing

Leverage the Desktop

- AMD, Intel, and NVIDIA all support Vulkan
  - Test on different GPU architectures
  - Some GPUs/drivers don't support features you might expect
- Mobile and Desktop code 99% the same
  - Window management will take different paths

Bleeding Edge

- Be prepared for driver issues
  - That can be worked around

#2: Distributing Vulkan

Dynamically Bind to Vulkan

Consider using vulkan_wrapper.cpp from samples

C++

```
void* libvulkan = dlopen("libvulkan.so", RTLD_NOW | RTLD_LOCAL);
if (!libvulkan)
    return 0;

// Vulkan supported, set function addresses
vkCreateInstance = reinterpret_cast<PFN_vkCreateInstance>(dlsym(libvulkan, "vkCreateInstance"));
vkDestroyInstance = reinterpret_cast<PFN_vkDestroyInstance>(dlsym(libvulkan, "vkDestroyInstance"));
vkEnumeratePhysicalDevices = reinterpret_cast<PFN_vkEnumeratePhysicalDevices>(dlsym(libvulkan,
"vkEnumeratePhysicalDevices"));


...
```

Vulkan and MultiAPK

XML

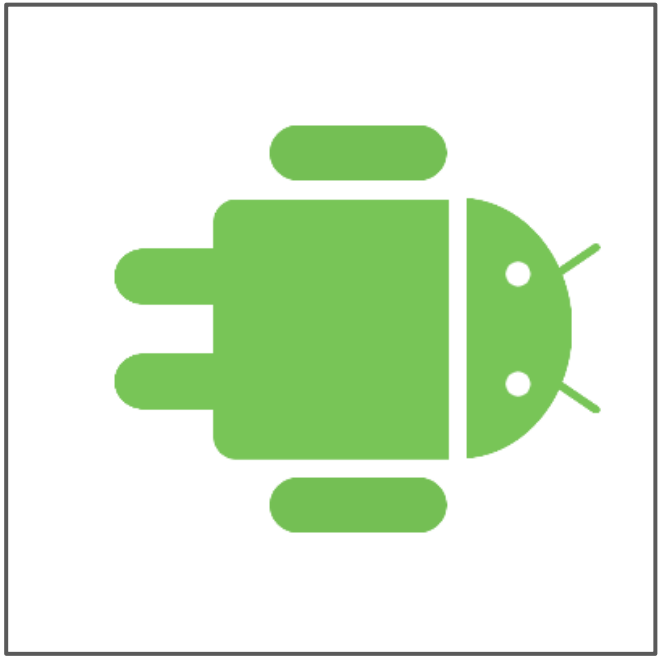
```
<uses-feature
    android:name="android.hardware.vulkan.version"
    android:required="true"/>
```

#3: Display Rotation

Apply Display Rotation During Rendering

Rotation Isn't Free on Android

- Android has a natural display orientation
- Letting the compositor rotate uses extra power/bandwidth
- Rotating while rendering the swapchain image is (essentially) free



Apply Display Rotation During Rendering

Rotation Isn't Free on Android

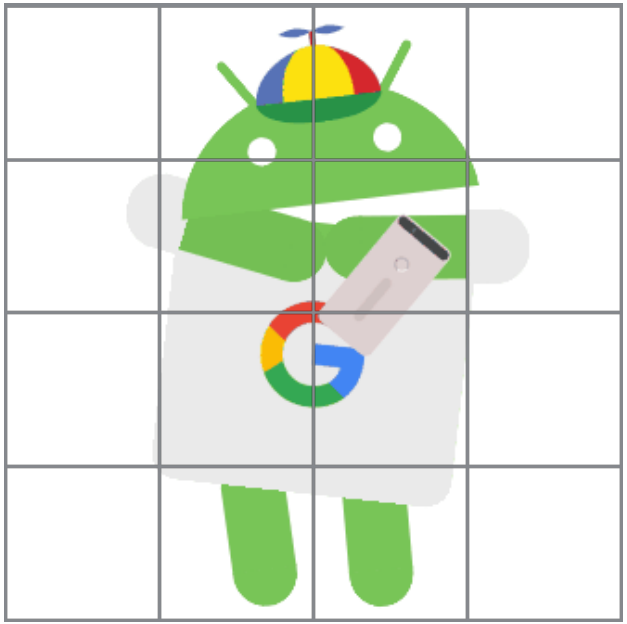
- VkSurfaceCapabilitiesKHR::currentTransform shows compositor rotation
- VkSwapchainCreateInfoKHR::preTransform is used to indicate to Vulkan that the app handles its own rotation

#4: Minimize Render Passes

Tile-based GPU Architecture

Most mobile GPUs are tile-based

- Render target split into separately rendered “tiles”
- Resources needed to render a single tile stored in on-chip memory
- Bandwidth for reading off-chip is limited



Immediate Renderers

You draw...

Triangle 1

then

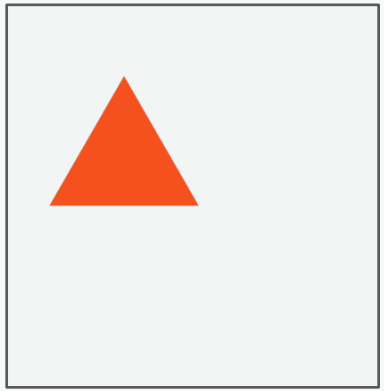
Triangle 2

then

Triangle 3

Google Developer Day

GPU Renders



Triangle 1

Triangle 2

Triangle 3

Tiling Renderers

You draw...

Triangle 1

then

Triangle 2

then

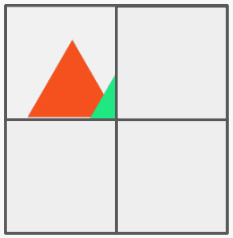
Triangle 3

Google Developer Day

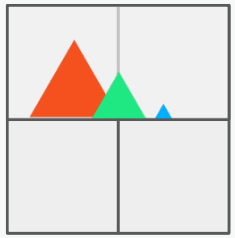
GPU Bins

Tri 1 Tri 2	Tri 2 Tri 3
Bin A	Bin B
Tri 2	Tri 2 Tri 3
Bin C	Bin D

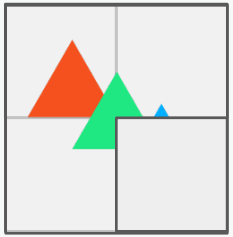
GPU Renders



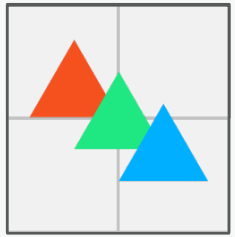
Tile A



Tile B



Tile C



Tile D

Multiple-Pass Rendering Techniques

Single-pixel Source Post-Process

- Tone mapping
- Exposure control
- HDR

Multi-pixel Source Post-Process

- Blur
  - Bloom
  - Focus control

Multiple-Pass Rendering Techniques

Deferred Shading

- Moving material data on-and-off chip in buffers
- Bandwidth & memory pressure can be painful

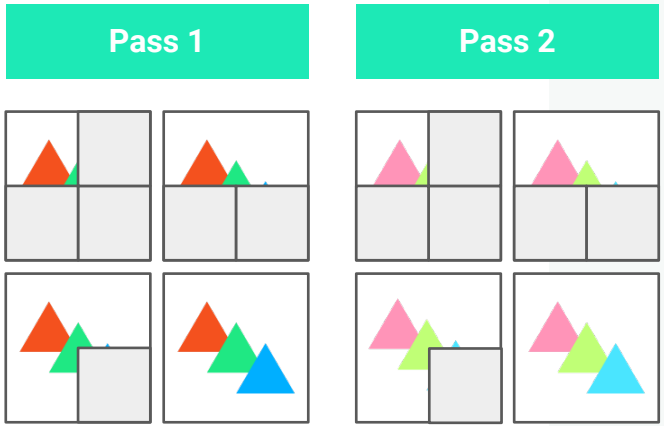
Volumetric Effects

- Participating media & accumulation
- Soft particles, depth-sample techniques

Multiple Passes on a Tiling Renderer

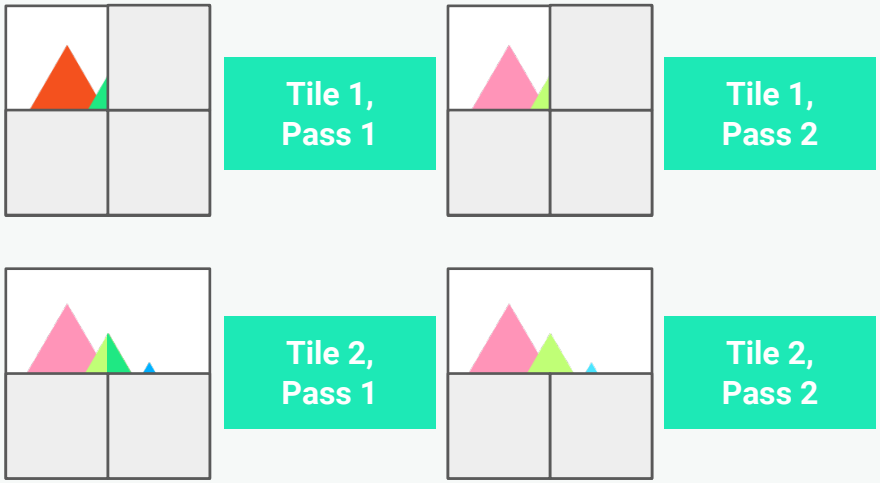
Without Vulkan Multipass

For each pass: render each tile and commit to memory. Later passes must reload. MSAA multiplies off-chip cost.



With Vulkan Multipass

For each tile: render each subpass. Relevant data stays on-chip. Resolve MSAA after last-pass, store once.



Tips for using subpasses

Attachments

- Set load/store disposition correctly
  - Load: VK_ATTACHMENT_LOAD_OP_DONT_CARE if you're overwriting entire buffer, VK_ATTACHMENT_LOAD_OP_CLEAR if you need it cleared
  - Store: VK_ATTACHMENT_STORE_OP_DONT_CARE if you're not using the contents again
  - Use VK_ATTACHMENT_LOAD_OP_LOAD and VK_ATTACHMENT_STORE_OP_STORE only when you actually need to load or store!
- Use transient attachments for never-stored data

Tips for using subpasses

Subpass dependencies

Specify **by-region**— otherwise it's a global dependency, and it waits on *all pixels* in prior pass

Single-pixel sources only (for now)

Each subpass can only access data for the same pixel in prior passes

Benefits

Efficiency, efficiency, efficiency

- Tiling renderers save precious clock cycles and power by avoiding extraneous read/write in low-bandwidth situation
- Opens the door for rendering techniques previously costly on mobile

- Immediate renderers can still benefit
  - E.g.: transient attachments never have to live anywhere outside of cache memory

#5: Memory Management

Choose Appropriate Memory

Applications Choose the Memory Type

- Different devices will have different memory types available
- Memory types are ordered so applications can use a simple algorithm to pick the best type for each use

- Mobile systems generally don't have separate physical memory heaps for CPU and GPU

  VK_MEMORY_PROPERTY_DEVICE_LOCAL_
  BIT is not as significant as it is on systems without shared memory.

Suballocate Device Memory

vkAllocMemory is Expensive

- Don't individually allocate small objects
- Alloc large blocks of memory, and then use a suballocator
- None Included - Expect this to be a common utility class

Effective Memory Management Strategies

Object Lifecycle tied to Rendering State

- Block allocate per-frame objects with a suballocator
  - Toss block into free heap after frames have rendered
- Use pools to avoid allocation during gameplay

vkDescriptorPool

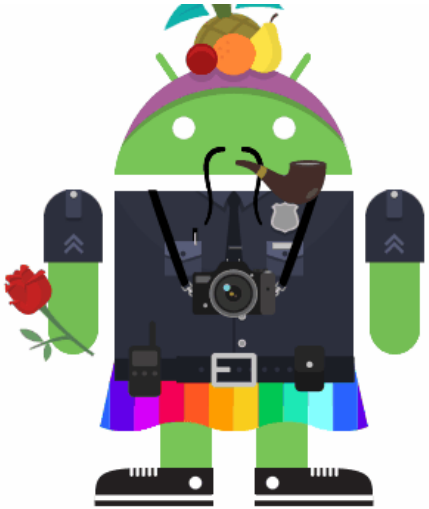
- Not consistently implemented by drivers
  - Consider rolling your own

#6: Batch API Calls

Batch API Calls

Vulkan does Batches!

- Perform multiple operations with a single call!
- Examples include vkQueueSubmit, vkAllocateCommandBuffers, vkUpdateDescriptorSets, vkCmdBindDescriptorSets



#7: Use Multithreading

Multithread!

Many Operations can be Parallelized

- Examples Include:
  - Loading textures and shaders
  - Building pipelines
  - Building command buffers
- Avoid Per-Thread Resources
  - Leads to Memory Bloat
  - Make sure to avoid cross-talk

- Threads are Heavy-Weight
  - Create a single-threaded path for debugging and for performance testing
  - Bad multithreading is worse than no multithreading

#8: Synchronization

Synchronization

With Great Power Comes Great Responsibility

Vulkan Includes 5 Primitives

- Fence
- Semaphore
- Event
- Memory barrier
- Pipeline barrier

Rendering to a Texture

C++

```
const VkImageMemoryBarrier barrier = {
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER, nullptr,
…   // lots of parameters here in usual Vulkan fashion
    image, {                                // VkImage image
        VK_IMAGE_ASPECT_COLOR_BIT,          // .aspectMask
        0, 1,                               // .baseMipLevel, .levelCount
        0, 1,                               // .baseArrayLayer, .layerCount
    },
};
vkCmdPipelineBarrier(cmdbuf,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT,  // srcStageMask
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT,          // dstStageMask
    0,                                  // VkDependencyFlags
    0, nullptr,                         // memory barriers
    0, nullptr,                         // buffer memory barriers
    1, &barrier);                       // image memory barriers
```

#9: Pipelines

Pipelines

Are Expensive to Create

- Pre-create if at all possible
  - Consider using dynamic analysis and tracing
  - Easiest way to drop frames and stutter

Pipelines and SPIRV-Cross

Enables Shader Reflection

- At Runtime
  - Determine Shader Resources
  - Query Push Constants

Enables GL Support for SPIRV Shaders

- Supports decompilation
- Emulation of Push Constants with uniforms

#10: Debugging

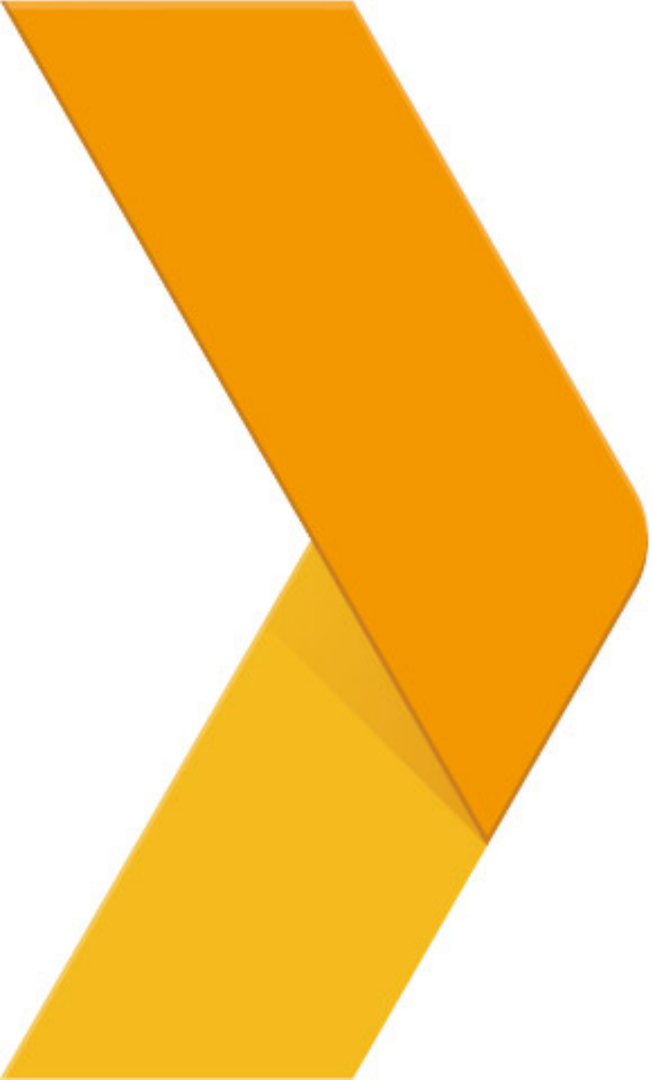
Capture/Replay Tools

VulkanTools

- Uses validation layers
  - vktrace
- Replay back on host
  - vkreplay

RenderDoc

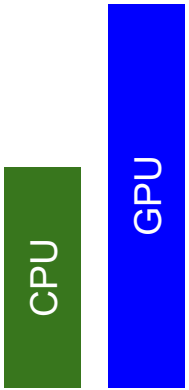
- Windows Only GUI
- Beloved Game development tool
- Requires modified APK



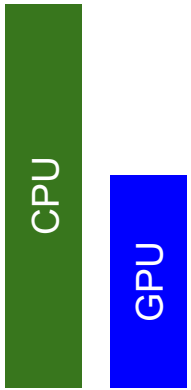
Should My Application Use Vulkan?

Vulkan Considerations

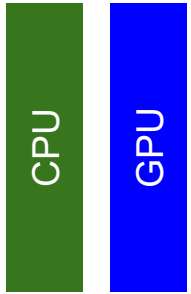
GPU Bound Rendering



CPU Bound Rendering



Thermally Bound Rendering



Vulkan in Game Engines

UNREAL ENGINE

4.14 Released

In Preview

Google Developer Day

In Conclusion

You Can Start Now

Vulkan is here and coming to more devices soon!

developer.android.com/ndk/downloads

github.com/LunarG/VulkanSamples

github.com/KhronosGroup/SPIRV-Cross

Google Developer Day

Google

Thank you!